PROCEEDINGS

55. IWK

Internationales Wissenschaftliches Kolloquium
International Scientific Colloquium

13 - 17 September 2010

Crossing Borders within the ABC

Automation,

Biomedical Engineering and

Computer Science

Faculty of
Computer Science and Automation

www.tu-ilmenau.de

TECHNISCHE UNIVERSITÄT
ILMENAU

Impressum
Published by

Publisher:	Rector of the Ilmenau University of Technology Univ.-Prof. Dr. rer. nat. habil. Dr. h. c. Prof. h. c. Peter Scharff
Editor:	Marketing Department (Phone: +49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de) Faculty of Computer Science and Automation (Phone: +49 3677 69-2860) Univ.-Prof. Dr.-Ing. habil. Jens Haueisen
Editorial Deadline:	20. August 2010
Implementation:	Ilmenau University of Technology Felix Böckelmann Philipp Schmidt

USB-Flash-Version.

Publishing House:	Verlag ISLE, Betriebsstätte des ISLE e.V. Werner-von-Siemens-Str. 16 98693 Ilmenau
Production:	CDA Datenträger Albrechts GmbH, 98529 Suhl/Albrechts
Order trough:	Marketing Department (+49 3677 69-2520) Andrea Schneider (conferences@tu-ilmenau.de)

ISBN: 978-3-938843-53-6 (USB-Flash Version)

Online-Version:

Publisher:	Universitätsbibliothek Ilmenau ilmedia Postfach 10 05 65 98684 Ilmenau

Home / Index:
http://www.db-thueringen.de/servlets/DocumentServlet?id=16739

MODEL-DRIVEN HMI DEVELOPMENT IN THE AUTOMATION INDUSTRY

Daniel Meisen, Sebastian von Klinski, Dragan Macos

Beuth Hochschule für Technik Berlin – University of Applied Sciences

ABSTRACT

The underlying article describes a case study for the usage of MDD in the automation industry. Together with an industry partner a MDD tool chain has been developed in order to create project-related HMIs for numerous application areas in the automation industry.

For the adoption of the MDD approach the overall system architecture has been adjusted. The underlying embedded applications provide a simple access interface to the input and output signals of the embedded application's function blocks. An embedded access layer runs in an additional low priority non-real-time process and provides a generic web-service interface to the embedded application.

On the client-side a generic client access layer implements access to the embedded application's signal values for the HMI client platform.

The MDD tool chain is used to graphically generate sophisticated HMIs for the client's automation products. Therefore, an Eclipse GMF graphical editor has been developed to create project-related dialogs from a universal and reusable control widget library.

Index Terms - MDD, HMI, SCADA, automation industry

1. INTRODUCTION

A few years ago an industry partner needed a new HMI (Human-machine interface) platform for their automation products that are mainly in the power conversion and automation industry. The HMI platform was to fulfill numerous requirements that are symptomatic for the ongoing changes in the automation industry. Shorter development cycles, an increasing diversity of hard- and software platforms as well as an increasing complexity of applications lead to a dramatic increase in development costs while the time pressure in the projects increased.

Due to the increasing use of off-the-shelf hardware, nowadays the automation hardware underlies only minor variations. In contrast, the applications and the necessary HMIs must cover a wide range of product- and customer-specific adjustments. Additionally, the application requirements and plant complexity increased continuously. HMIs, in turn, are supposed to encapsulate the underlying complexity with easy-to-understand graphical user interfaces.

Aim of the underlying project was to develop a MDD infrastructure for the graphical modeling of HMIs. With this approach, development costs were to be reduced while quick adjustments to the HMIs were to be facilitated. Additionally, HMIs should be implemented by non-technical experts to reduce the development load of the technical experts.

2. OVERALL ARCHITECTURE

The systems used in the automation industry are controlled and monitored using embedded devices (Figure 1) that can be accessed by a SCADA (Supervisory Control and Data Acquisition) system. In the past, these devices consisted mainly of custom hardware, specific for every single type of system. A large amount of engineering effort had to be spent to develop and test these custom hardware components. Most of the application logic of those systems has been implemented by the system's hardware.

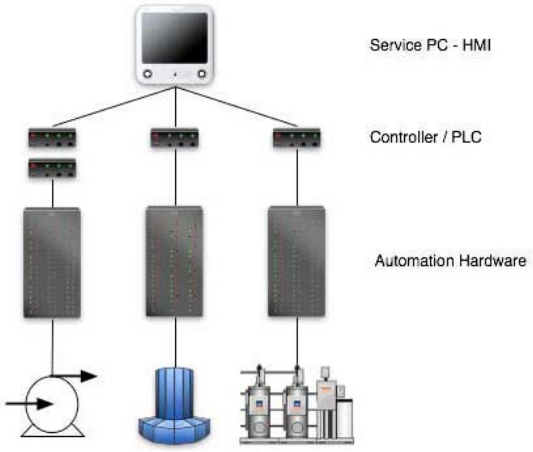

Figure 1: typical automation product components

In recent years powerful off-the-shelf embedded hardware has become available. This changed the general development process for applications in the automation industry. Nowadays, most parts of a system's application logic are implemented in software, running on standard industrial grade off-the-shelf hardware.

Off-the-shelf hardware (e.g. industrial grade automation PCs) can be used in a variety of different automation products without major hardware-modifications. These devices provide different kinds

of interfaces, such as Ethernet-, CAN- or RS232-ports, specific to the type of system and can be extended via additional PCI/PCI-Express interface cards.

The automation PC, i.e. the controller, is used to monitor and control other hardware components of an automation system. Common hardware components are sensors and actuators.

The controller's application data will be accessed by a SCADA-system that collects the signal values and renders complex graphical HMIs based on the data. The main component of a SCADA system is the HMI that allows workflow visualization and manipulation that is specific for each automation system or system type.

The diversity and complexity of the software applications running on these controllers has increased constantly requiring a new overall system-architecture (Figure 2).

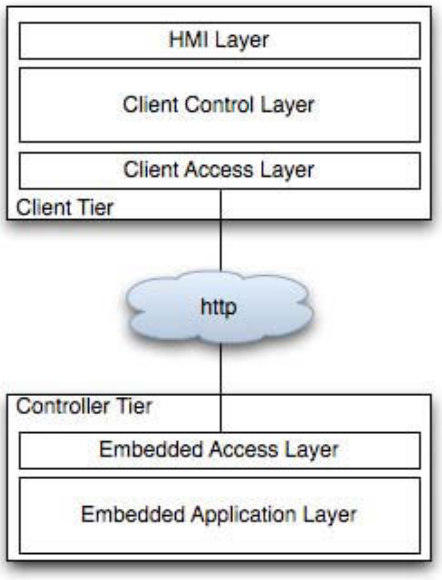

Figure 2: Overall Application Architecture

This architecture is similar for all of the client's automation products and basically consists of a controller tier with an embedded application layer – the system's control application – and an access layer. The access layer allows to read the embedded application's output-signals and to modify the values of the application's input signals.

For the user-friendly display and modification of the embedded application's signal values a sophisticated HMI is implemented in the client tier. The client tier consists of three layers:

- A client access layer, implementing the access interface to the controller's embedded application
- A client control layer implementing components for the lifecycle management of a controller's state and the embedded application's signal values
- A HMI layer that implements a runtime component for graphical HMI diagrams as well as set of list-based views for common embedded application information

Thus, the overall application architecture can be subdivided into a client and a controller tier (Figure 2).

2.1. Controller Tier

The controller tier runs on the controller hardware, i.e. the automation PC, and implements two different layers: an embedded application layer, comprising the system's embedded application and an additional embedded access layer that provides a remote access to the embedded application's signal values.

2.1.1. Embedded Application Layer

An embedded application is designed from a set of highly customizable logical components (so called function blocks) that have a number of input and output ports that are connected via signals (Figure 3). A signal has a specific data type, a unique address within the embedded application and a value.

The value of an output-signal is computed within a function block; the value of an input signal can be either set manually or derived from the output of the application's function blocks.

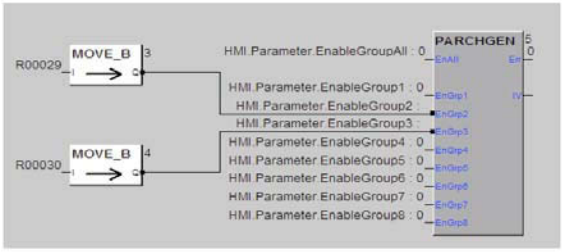

Figure 3: Function Block Diagram

2.1.2. Embedded Access Layer

The embedded access layer implements an interface to the embedded application layer that can be accessed by the client tier via web services. An embedded HTTP server runs in a low priority non-real-time process next to the embedded application's processes and provides a web-service interface that can be accessed via standard protocols such as http.

A range of different message-oriented web services provide the access to the embedded application's input and output signals. Manipulation of values of single input signals or sets thereof is also supported.

An additional web application that runs within the controller tier's web server can be used for very basic manipulation of input-signal values and the retrieval of the most basic controller information using a web-browser.

2.2. Client Tier

The client tier is implemented as a multi-platform rich client application based on the Eclipse RCP-Framework [1]. The client can be used to monitor and control a single controller or groups thereof (so called plants). In some projects more than 20 controllers are addressed simultaneously by one HMI.

The client continuously monitors the controller's state and can display the state of the controller and the embedded application's information. A very rudimentary display mode provides a list of all input and output signals as well as error- and event-log messages in generic list views.

For more sophisticated and customized HMI interfaces the MDD tool chain is used. Therefore, the HMI client includes a model-based graphical editor and a runtime component. The graphical editor is used to create new HMI component descriptions. The runtime component is used to execute the graphical HMI diagram descriptions as normal views in the client.

The custom HMI components of the client basically consist of three different layers:

- A client access layer, that implements the communication between the client- and the controller tier
- A client control layer, that manages the controller's states and offers additional services e.g. for storing and recording controller information or the values of the embedded application's signals
- A HMI layer that implements the graphical interaction with a controller through list views and graphical user interface diagrams (see Figure 4).

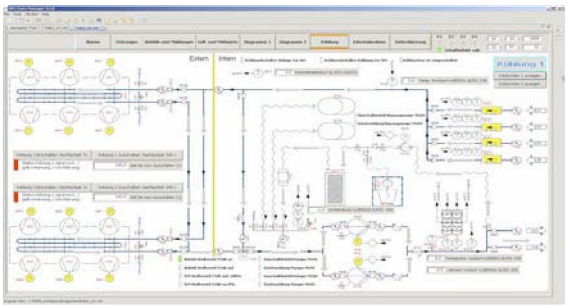

Figure 4: Graphical HMI

2.2.1. Client Access Layer

A controller's embedded application is addressed using the controller's specific IP address. During the handshake between the controller and the client the controller sends a description of all input and output signals of the embedded application. This controller description is interpreted by the client and can be referenced by the client control layer.

The client access layer is communicating with the controller tier via a variety of web services provided by the controller. The client tier uses the web services to request the values of the input and output signals, the controller's alarm- or event-message log or other controller specific information.

These web service requests can be accessed from the client control layer using a simple API.

2.2.2. Client Control Layer

The client tier is able to monitor multiple controllers or groups thereof simultaneously. A client control layer manages the state of all connected controllers.

A watchdog thread continuously monitors the controller's application and connection state and triggers application events if a state change occurs, e.g. a controller is disconnected.

The values of the embedded application's signals are retrieved periodically by a scheduler component and can be visualized using generic list-views or graphical HMI diagrams.

2.2.3. HMI Layer

Often control systems require complex visualizations of the embedded application's input and output signals, workflow processes, or system states. Additionally, GUI-based manipulation of the application's signal values must be provided to allow user manipulation of the embedded steering processes. Those GUIs usually require user adjustments for a specific automation product. This often includes visual process animations as well as adjusted display of the embedded application's input and output signal values.

In addition to the display of an embedded application's signal values in various list-based views the HMI layer implements a runtime component for the rendering of system-specific GUIs. This runtime component is a runtime platform for the graphical HMI diagram components that are described in a proprietary description format.

A graphical HMI diagram is developed by arranging a selection of graphical control widgets from the library of system control widgets on various pages using the client's graphical modeling component. The resulting graphical definition of the graphical HMI diagram can be exported into a XML-based HMI description format.

2.2.4. HMI Description Format

The HMI description format is based on XML and describes the components that comprise a graphical HMI diagram, the component's arrangement and their layout. A graphical HMI diagram is described by a set of control widgets that are arranged on pages. The visual appearance of the control widgets is defined by additional layout (e.g. size and position) and styling information (background or foreground color, fonts).

The HMI description format also contains information that is necessary to connect the control widget to the input and output signals of a controller's embedded application such as a signal's address and a controller's IP address.

The control widgets allow the creation of complex animated HMI visualizations for a variety of different applications using a large control widget library. These animations can be assembled from custom, application specific images. The description format therefore contains references to all resources that are referenced from a graphical HMI diagram's control widgets.

3. MDD-APPROACH

The developed MDD tool chain consists of a graphical modeler (Figure 5), a graphical HMI description format generator and a graphical HMI diagram runtime component.

The graphical modeler is based on the Eclipse frameworks (i.e., GMF [2], EMF [3], GEF [4], etc.). The underlying DSL (domain-specific language) is defined using EMF.

For the transformation of the EMF model into the HMI description format a proprietary XML-based code generation approach is adopted. The HMI description format contains both the description of the graphical HMI diagrams as well as the specification of the access to the according input and output signals of the specific embedded application.

The HMI descriptions are interpreted and executed by the HMI runtime component. This HMI runtime creates the actual GUI components using SWT and JFace widgets and connects those widgets with the according embedded backend using an adjusted widget controller. This widget controller directly accesses the embedded application using adjusted web-service calls to the embedded-tier access layer.

3.1. HMI Modeling

Graphical HMI diagrams are divided into multiple pages, which in turn comprise the control widgets. In the first step, EMF has been used for the specification of the DSL. It allows the description of the HMI pages together with the contained control widgets. In the DSL all control widgets provided by the platform have been specified with the required access interface to the embedded application. With this EMF model a graphical editing tool has been generated using the Eclipse Frameworks GMF and GEF.

The palette of the GMF modeling tool contains all available widgets (see Figure 5).

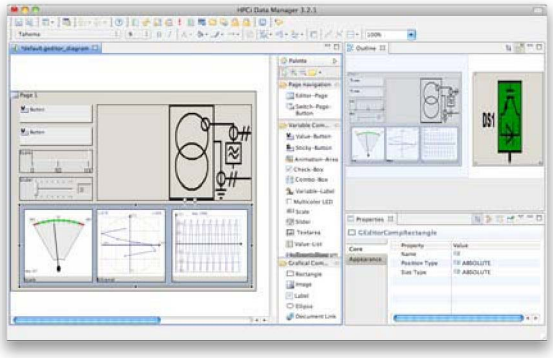

Figure 5: Screenshot of the HMI Modeling Tool

The provided system control widgets comprise passive and active components. Passive components are not associated with any of the embedded application's signals and can be used for structuring and decorating the graphical HMI diagram pages. Passive widgets include labels, images and various kinds of geometrical figures (e.g. rectangles, ellipses).

Additionally, there is large range of active control widgets that have to be associated with one or more signals of an embedded application. These active control widgets can be subdivided in display and manipulation widgets.

Display widgets allow the display of input- and output-signal values or a derived value. Commonly used are labels that display output values and images that are adjusted in their appearance according to the value of an output signal, thus allowing animations based on the value of the controller's embedded application signals (Figure 6).

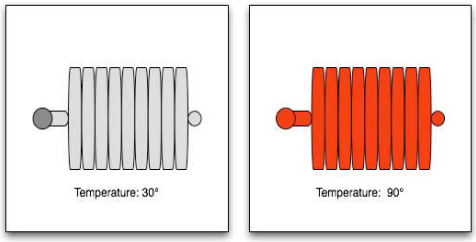

Figure 6: Animation Area

Manipulation widgets offer graphical controls to alter the value of an input signal's value. A comprehensive set of manipulation control widgets is provided for manipulating the values of the embedded application's input signals. Buttons, sticky buttons, check-boxes, combo-boxes, scales, and several other common widgets can be added to an HMI page to manipulate arbitrary input values with varying data types.

Finally, there is an increasing set of sophisticated display components that allow the creation of rich graphical HMI diagram pages. Time-dependent trend diagram widgets, bi-signal diagram widgets (that display the history of two signal's values) or adjustable dynamic function diagram widgets - that

can be used to display multiple points or curves based on signal values or mathematical derivations thereof (Figure 7) - are some of the more complex display components provided by the widget library.

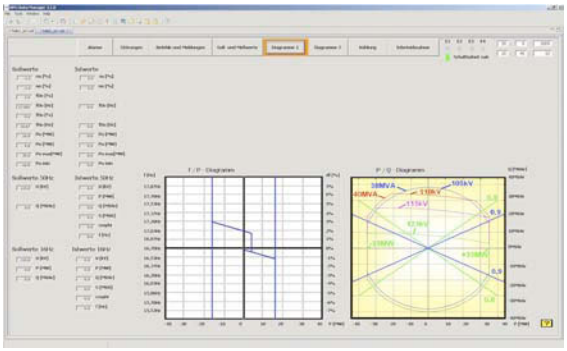

Figure 7: Function Diagram Widget

The graphical HMI modeler includes user-friendly dialogs to manage the connections between the control widgets and the embedded application signals. In order to connect the control widget to the embedded application's signals a connection to the controller's web service is established by utilizing the client control layer.

An arbitrary number of controllers can be integrated in one graphical HMI diagram. Each active component is attached to the signal of one or more signals of a controller's embedded application using the IP address of the controller and the application's input- or output-signal address. The description of components therefore usually comprises the specification of the location, the horizontal and vertical sizes, as well as the controller signal specification.

3.2. HMI-Description Generation

The modeled HMI description is initially stored in the EMF format that includes several EMF specific entries. For the serialization of the EMF files the XML Metadata Interchange format (XMI) is used. The XMI format has been specified by the OMG Group and is commonly used as an exchange format for development tools. The HMI description for the target controller is generated by transforming the EMF model into a proprietary XML-based HMI description format. Therefore, the EMF model is parsed, exporting all relevant information like component positioning and alignment, layout, as well as access to the embedded application's signals.

A proprietary exporter component has been implemented to optimize the integration of the HMI generation into the productive HMI client. Thus, new HMIs can be used immediately after the modeling step. The exporter component is conceived in a way that it can easily be extended by new GUI components. Each model element is mapped to a SWT widget or a group of widgets. The exporter then retrieves an according producer class using a factory, where all producers are registered. The producer in turn generates the according HMI description for the currently exported HMI component.

The HMI modeling chain has been conceived to allow a simple integration of new HMI components. By registering a new producer at the factory new component types can be added to the HMI modeling chain. As HMIs are highly dependent on project-related requirements this extensibility of the HMI modeling chain is indispensible to ensure a long-term usage and avoid software ergonomic limitations.

The HMI description is a simplified version of the EMF model and is limited to the data, required for rendering the actual runtime HMI. The specification of the embedded application access is based on the specific application design with respect to the input and output addresses.

The actual IP addresses of the controllers are added to the HMI description during the modeling process. They can still be modified during the setup of the productive HMI client in order to allow the re-usage of diagrams for several controllers. The correct mapping of the input and output signals between the controllers has to be ensured in this case, though. Therefore, signal names have been introduced to allow a mapping of names to actual hardware addresses of signals. This concept allows switching applications and diagrams between controllers without the need to adjust hard-coded signal addresses.

Resources such as images or animations are bundled together with the HMI description making a graphical HMI-diagram easily distributable as a single jar file.

3.3. HMI Runtime

The client tier is implemented as an Eclipse RCP application. The actual graphical HMI diagrams are Eclipse views. They are registered as extensions to the org.eclipse.ui.views [5] extension point. The pages are implemented as individual composites in a stack layout. The various control widgets are created mostly using standard SWT [6] and JFace [7] widgets. Control widgets that display graphical animations or diagrams are implemented by custom widgets mostly based on Draw2D [8].

When the HMI runtime is started a request scheduler is created that manages the controller(s) state and coordinates the retrieval of signal values by periodically requesting the controller(s) signal values through the client's access layer.

The HMI can consolidate the responses from an arbitrary number of controllers. To minimize network traffic the requests to the controllers are consolidated by the request scheduler. Usually, only the values for signals that are visible from the HMI client's views are requested from the controller. However some control widgets display a history of signal values over

time. Signals that are connected to such a history control widget will be requested constantly, even if they are on a diagram page that is not currently visible.

4. RESULTS

The MDD tool chain has been applied to a large variety of different kinds of productive automation products such as solar plants, heavy steel production sites, wind energy plants, power converters, and various other application areas. In these application areas the MDD tool chain generated rich graphical HMI diagrams that can directly interact with a controller's embedded application utilizing the client's control and access layers through web-services.

Due to the graphical modeling approach the HMI generation could be delegated to non-technical team members and functional experts. Additionally, HMI development expenses and development time to create HMIs for specific products could be reduced.

Overall, the MDD approach helped the industry partner to cope with the new industry challenges described above and significantly shortened the development-cycle for their HMIs.

4.1. Extensibility

An additional feature of the described HMI architecture is the possibility of extending the control widget library. Adding a new control widget can be performed in three steps: (1) the component has to be added to the clients HMI EMF-model; (2) a producer-component for the generation of the control widgets HMI description format has to be implemented; and (3) a runtime representation of the widget has to be implemented using standard SWT or JFace controls or utilizing Draw2D.

4.2. Cost Reduction

The HMIs are currently running on additional panel PCs or separate service PCs that are not part of the automation system itself. To further reduce costs for the HMI, the usage of low cost embedded touch panel PCs is being evaluated. This can be achieved by either running the HMI application on the systems embedded controller tier using the Eclipse RAP (Rich Ajax Platform [9]) project. To access the HTML/AJAX based UI – rendered from the original SWT/JFace-UI – the low cost embedded touch panel PC only needs to have a web-browser installed (Figure 8).

Another option is to use a slightly modified version of the HMI based on the eRCP (embedded Rich Client Platform [10]) running directly on the low cost embedded touch panel PC. Compared to the Eclipse RCP the requirements for eRCP (e.g. Java Profile) are more likely to be fulfilled by a low cost embedded touch panel PC. The limited performance of the low cost touch panel would still allow the client to contain list-based views and the graphical HMI runtime component. However the graphical modeler would not be included.

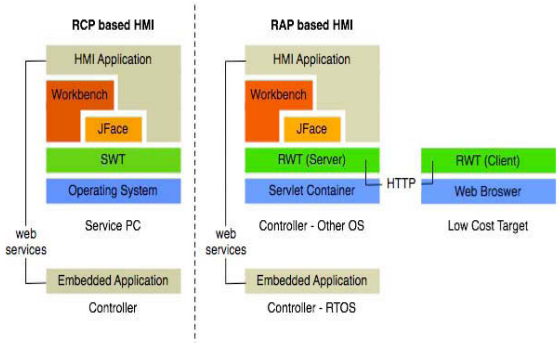

Figure 8: RCP / RAP Architecture

5. REFERENCES

[1] Eclipse Rich Client Platform, http://wiki.eclipse.org/index.php/Rich_Client_Platform, The Eclipse Foundation

[2] Graphical Modeling Framwork, http://www.eclipse.org/modeling/gmp/, The Eclipse Foundation

[3] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/, The Eclipse Foundation

[4] Graphical Editing Framework, http://www.eclipse.org/gef/, The Eclipse Foundation

[5] Platform Extension Points, http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/extension-points/index.html, The Eclipse Foundation

[6] SWT: The Standard Widget Toolkit, http://www.eclipse.org/swt/, The Eclipse Foundation

[7] JFace, http://wiki.eclipse.org/JFace, The Eclipse Foundation

[8] Draw2D Toolkit, http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.draw2d.doc.isv/reference/api/org/eclipse/draw2d/package-summary.html, The Eclipse Foundation

[9] Rich Ajax Platform, http://www.eclipse.org/rap/, The Eclipse Foundation

[10] Embedded Rich Client Platform, http://www.eclipse.org/ercp/, The Eclipse Foundation